Database CRM

v. 1.0

***Created by Svitlana Korostelova***

***(group B19, Vinnytsia IT-Academy)***

**Task obtained:** 07.03.2020

**Pre-defence:** 21.03.2020**, Teacher1\_\_\_\_\_\_\_\_\_\_\_**

**Defence:** 28.03.2020**, Teacher2\_\_\_\_\_\_\_\_\_\_\_**

**CONTENTS**

**1. System requirements and instructions for deploying the program**

**2. About the typical destination the database CRM and specifics of use**

**3. Description of the main variables and main functions**

**1. System requirements and instructions for deploying the program**

Operating system should be macOS Mojave(10.14) or higher.

Open <https://gitlab.intita.com/svitlana.korostelova/DataBaseCRM/>, build and run it with Xcode program. For deep testing, you can use also any [Database Viewer](https://soft.mydiv.net/win/download-Database-Viewer-Editor.html) (even online) which supports sqlite3 files.

**2. About the typical destination the database CRM and specifics of use**

The database was created for accounting by the manager of his work. A client event table is attached to the main client table. The user does not have the right to upload records to the table or delete them in any order. It is only allowed to add entries manually after the event because they indicate the manager’s work (when and how many times he called the client, signed documents or accepted payment). Relationship between DB tables by client ID (unique by auto-increment in SQL). Implemented cascading deletion of client events if its removal from the database.

For the initial loading of data into DB SQL, the project folder contains a template with Latin letters of convenient length for the correct appearance and analysis of reports in the console. DB also works freely with the Cyrillic alphabet without limiting the size of the entries entered by the user. There are requirements for incoming information: after each line with information there should be a separator "|" (after the title, contactPerson, contactInfo field).

The functionality of this program allows you to add, edit, delete customer cards, add them a lot of events, view the various reports, and also backup the database to a file. For using the program just switches the navigation menu according to the displayed hints.

At all stages of the user’s work with the program established the control over the entry of incorrect data. If you inadvertently enter incorrect information, you do not need to restart the program, an error message will be displayed and will be an automatic return to the previous navigation menu step.

**3. Description of the main variables and main functions**

The program is one module, consisting of nine functions with the algorithms in separate ".c" files and one ".h" file, which describes the main global variables and function prototypes.

**EventType** - the enumeration, which provides main actions you can interact with client. **EVENT\_TYPE\_STRING** - the array of strings with text match to EventType enumeration.

**struct Customer** - main client structure with integer "id" field. It contains arrays of strings: "title", "contactPerson", "contactInfo".

**struct Event** - the structure, related with the main client structure. It contains the integer "id" field, struct "customer", enumeration "EventType" and an array of strings "description".

**extern** **struct Customer \*customers** - global array of structures. At the moment in the programm dynamically memory is allocated to 200 such structures (clients). **extern** **int customersCount** - global variable for counting steps during getting around the array of structures "customers".

**extern** **struct Event \*events** - global array of structures. At the moment in the programm dynamically memory is allocated to 400 such structures (events). **extern int eventsCount** - global variable for counting steps during getting around the array of structures "events".

**accessLoadDataFromFile()** - Checks the permission to download information by a package to the database (for this you need to know the administrator password). If the password is correct, it calls the function downloadClientsBase.

**downloadClientsBase()** - Accepts the file name from the user (where the information will be loaded from), parse it line by line in a loop using address pointer arithmetic, and also calls insertCustomerToSQL line by line to write data to the database.

**addCustomer()** - Accepts the details of the new client from the user and writes to the database, calling insertCustomerToSQL.

**removeRecord()** - Deletes a client record from the SQL database. Surely deletes an entry in the Events table associated with it by a unique ID.

**redactRecords()** - Gives the user to select the field that needs to be changed in the database and gives it into function updatingRecord.

**updatingRecord()** - Writes changes from redactRecords to SQL by match the field received from the user with the database table field.

**addNewEvent()** - Gives the user the opportunity to select predefined events from the enumeration, enter a description of the event, then writes a row into the SQL database to the Events table, contacting by a unique client ID.

**uploadClientsBase()** - Unloads the contents of the Customers table into a file, separating the fields with the "|"

**insertCustomerToSQL()** - It writes the parameters of the new client to the database according to the correspondence of the fields received from the user with the fields of the database table. Also writes in a loop a batch of clients on the first boot.

**readCustomers()** -Updates the full client structure from the SQL database into the calling place. Also calls resetCustomers - reloading the array of Customer structures, dynamically allocating memory for new work with it. readCustomers is needed when, after previous user actions (for example, selection by word), incomplete information is stored in the array of Customer structures and needs to be updated. Calls in a loop for each row from the SQL database readCustomersCallback.

**readCustomersCallback()** - Walks through in a loop for each row from the SQL database (customers table) and reads for the current row: in **argc** - the number of columns in the database; in **\*\*argv** - as many values ​​as columns for the current row in the database table; in **\*\*azColName** - arrays of characters consisting of database column names.

**readEventsCallback()** - Walks through in a loop for each row from the SQL database (events table) and reads for the current row: in **argc** - the number of columns; in **\*\*argv** - values in the current row for each column; **\*\*azColName** - column names.

**resetEvents()** - Dynamically allocates memory for an array of Events structures, sets the counter of this array to zero.

**resetCustomers()** - Updates the array of Customer structures, dynamically allocating memory for new work under it and resets the counter by the array.

**getCustomerById()** - Returns a struct Customer element by its id. If the client is not in the database, it will return NULL to the place of the call, then the user will be given a message about the absence such id.

**showReports()** - Includes 4 key reports of the database. **readCustomersForOneEvent()** - Shows a report about all clients for one event, chosen by user.

**readEventsForOneCustomer()** - Shows a report on all events for one client.

**selectingByPartOfWord()** - Shows a report for any value entered by the user (for example, part of a name or phone). Searches for the entered character set in the SQL database (customers table) and displays all rows where there are fields with the match to those character set.

***Copyright © 2020 Svitlana Korostelova. All rights reserved.***